Constant-Overhead Secure Computation using Preprocessing

Ivan Damgård, Sarah Zakarias

Aarhus University, Denmark

Multiparty Computation

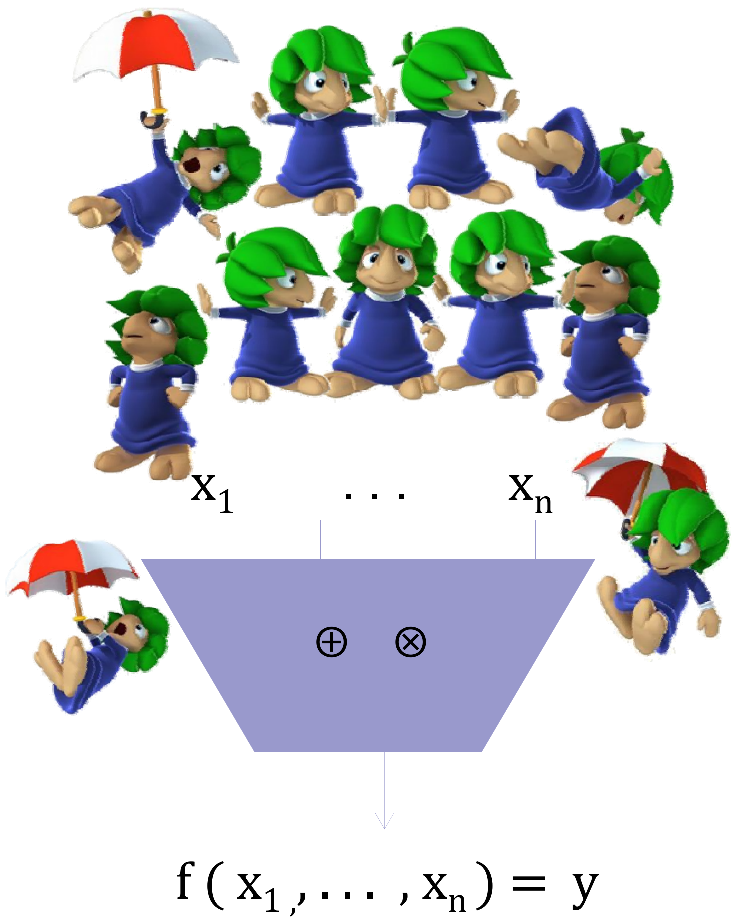
Goal: Compute circuit UC-securely

Unlike previous talk:
Interested in complexity of protocol when circuit size grows

$$f(x_1, \ldots, x_n) = y$$

MCP Flavour in this talk

- **Dishonest Majority**
  - *N* players, up to *N -1* corrupted
  - No info. theo. sec. from scratch
  - Need pk-encryption
  - No termination guarantee
  - Natural model for 2-party case



- **Boolean Circuits**

Preprocessing Model

Online phase (our protocol)

- Assume trusted dealer providing ‘raw material’
- Use only cheap information theoretic primitives
- Evaluate circuit given inputs

Preprocessing (not this talk)

- Implement trusted dealer (independent of circuit/inputs)
- Use public-key techniques
- Run any time prior to the computation

A couple of notions

Preprocessing model

- ***Universal.*** No knowledge about circuit nor inputs
- ***Dedicated.*** Circuit known but inputs unknown

Overhead for on-line phase
(how much resource per player per gate)

- ***Data.*** Total number of bits to store divided by $N \cdot |C|$
- ***Communication***. Communication complexity divided by $N \cdot |C|$
- ***Computation.*** Computational complexity divided by $N \cdot |C|$

Previous Work in Preprocessing Model

[Damgård, Pastro, Smart, Zakarias 12]

[Damgård, Ishai, Krøigaard 10]

[Nielsen, Nordholt, Orlandi, Burra 12]

For large fields F ($|F| \approx 2^k$, k is security parameter), overheads are O(1)

For small fields, overheads are $\Omega(k)$ or N polylog(k) log(|C|).

- Can we get O(1) overhead also for small fields, say F_2?

Our Results

There exists an N-party protocol in the preprocessing model for computing a Boolean circuit C *statistically secure* against *N -1* active corruptions.

For error probability 2^{-k} the overheads are:

- $O(1)$ data and communication, and $O(1 + k/N)$ computation in the dedicated preprocessing model
- $O(\log(|C|))$ data/comm, and $O(\log(|C|)\,(1 + k/N))$ computation in the universal preprocessing model

What can we hope for?

- In [DPSZ12], lower bound: data and computational overhead for universal preprocessing must be $\Omega(1)$.
- Bound for data overhead holds also for dedicated preprocessing.
- Intuition suggests that computation overhead should be $\Omega(1)$ in general.
- [Ishai et al 13]: Subconstant data *and* communication overhead would require breakthrough in PIR protocols.

So: from current knowledge, $O(1)$ overheads seems to be the best we can realistically hope for.

Some basic (known) ideas

[DIK 10] Can assume we evaluate circuit by blockwise computations:

$$\mathbf{x} + \mathbf{y} = (x_1, \ldots, x_n) + (y_1, \ldots y_n) = (x_1 + y_1, \ldots , x_n + y_n)$$

$$\mathbf{x} * \mathbf{y} = (x_1, \ldots, x_n) * (y_1, \ldots y_n) = (x_1 \cdot y_1, \ldots , x_n \cdot y_n)$$

[DPSZ 12] Authenticate with global key and secret share

$\mathbf{x}^1, \mathbf{m}^1 \in \{0,1\}^n$

$$\mathbf{x} = \mathbf{x}^1 + \mathbf{x}^2$$

$$\mathrm{MAC}(\mathbf{x}) = \boldsymbol{\alpha} * \mathbf{x} = \mathbf{m}^1 + \mathbf{m}^2$$

Global secret key

$\mathbf{x}^2, \mathbf{m}^2 \in \{0,1\}^n$

Combining Ideas

Problem: Too easy to cheat with 1-bit MACs!

Authenticate with global key and secret share

$\mathbf{x}^1, \mathbf{m}^1 \in \{0,1\}^n$

$\mathbf{x} = \mathbf{x}^1 + \mathbf{x}^2$

$\mathrm{MAC}(\mathbf{x}) = \boldsymbol{\alpha} * \mathbf{x} = \mathbf{m}^1 + \mathbf{m}^2$

$\mathbf{x}^2, \mathbf{m}^2 \in \{0,1\}^n$

Combining Ideas

Problem: Too easy to cheat with 1-bit MACs!

Solution: Good Linear Error Correcting Code C

$C(\mathbf{x}) \in \{0,1\}^n$ is encoding of $\mathbf{x} \in \{0,1\}^k$ in C

Authenticate with global key and secret share

$C(\mathbf{x}^1)$, $\mathbf{m}^1$
$\in \{0,1\}^n$

$$C(\mathbf{x}) = C(\mathbf{x}^1) + C(\mathbf{x}^2)$$

$$\begin{aligned} MAC(C(\mathbf{x})) &= \boldsymbol{\alpha} * C(\mathbf{x}) \\ &= \mathbf{m}^1 + \mathbf{m}^2 \end{aligned}$$

$C(\mathbf{x}^2)$, $\mathbf{m}^2$
$\in \{0,1\}^n$

Authentication based on Linear Codes

message $C(\mathbf{x}) \in C$

$m(\mathbf{x}) = \alpha * C(\mathbf{x})$
(many 1-bit MACs in parallel)

$C(\mathbf{x})$ →

$C(\mathbf{x}) + \mathbf{e}$ ←

$\mathbf{e'}$ ←

<u>Check:</u>

- $m(\mathbf{x}) + \mathbf{e'} = \alpha * (C(\mathbf{x}) + \mathbf{e})$
- $C(\mathbf{x}) + \mathbf{e}$ is a codeword

Adversary wins if:
$\mathbf{e} \neq \mathbf{0}$ & check is OK

***e** must be a codeword*
⇒adversary must cheat in many positions to win.

Secret Representation

$$C(\mathbf{x}) = C(\mathbf{x}_1) + C(\mathbf{x}_2)$$

$$m(\mathbf{x}) = \boldsymbol{\alpha} * C(\mathbf{x}) = m(\mathbf{x})_1 + m(\mathbf{x})_2$$

$C(\mathbf{x}_1), m(\mathbf{x})_1$ $\qquad$ $C(\mathbf{x}_2), m(\mathbf{x})_2$

$[\mathbf{x}]$

- $\boldsymbol{\alpha}$ generated in preprocessing, will be released as needed
- Cannot check MACs during protocol ($\boldsymbol{\alpha}$ known → forgery)
- ***Partial openings*** : open shares, check valid codewords but postpone checking of MACs

Computations

Sum of [x] and [y]

- Locally & component-wise

Problem: the product of two codewords is not a codeword!

$C(x_1) + C(y_1)$ $\quad$ $C(x_2) + C(y_2)$

$m(x)_1 + m(y)_1$ $\quad$ $m(x)_2 + m(y)_2$

[x + y]

Multiplication of [x], [y]

- Beavers Circuit Randomization

  - Preproc. gives random [**a**], [**b**], [**c**] st. **c** = **a** * **b**

  - Open **ε** = C(**x-a**) = [**x**] – [**a**], **δ** = C(**y-b**) = [**y**] – [**b**]

  - Compute [**x*y**] = [**c**] + **ε** * [**b**] + **δ** *[**a**] + **ε** * **δ**

Linear Codes - now with multiplication

- C: [*n*, *k*, *d*] linear code, length n, dimension k, min. distance d
- $C^* := \{\mathbf{c} * \mathbf{c'} \mid \mathbf{c}, \mathbf{c'} \in C\}$ is the Schur-transform of C
- C^* : $[n, k^*, d^*]$ linear code with $d^* \leq d$, and $k^* \geq k$
- $C^*(\mathbf{x})$:= codeword in C^* where **x** appears first
- $C(\mathbf{x}) * C(\mathbf{y}) = C^*(\mathbf{x} * \mathbf{y})$
- Asymptotically good constructions with different trade-offs using Reed-Solomon or Algebraic Geometry Codes [CCX11]

Computations

Linear Computations

- Locally & component-wise

Multiplication by codewords introduce vectors in C*.

$C(\mathbf{x}_1) + C(\mathbf{y}_1)$
$m(\mathbf{x})_1 + m(\mathbf{y})_1$

$C(\mathbf{x}_2) + C(\mathbf{y}_2)$
$m(\mathbf{x})_2 + m(\mathbf{y})_2$

$[\mathbf{x} + \mathbf{y}]$

Multiplication

- Beavers Circuit Randomization
  - Preproc. gives random $[\mathbf{a}]$, $[\mathbf{b}]$, $[\mathbf{c}]^*$ st. $\mathbf{c} = \mathbf{a} * \mathbf{b}$
  - Partially open codewords $\boldsymbol{\varepsilon} = [\mathbf{x}] - [\mathbf{a}]$, $\boldsymbol{\delta} = [\mathbf{y}] - [\mathbf{b}]$
  - Compute $[\mathbf{x} * \mathbf{y}]^* = [\mathbf{c}]^* + \boldsymbol{\varepsilon} * [\mathbf{b}] + \boldsymbol{\delta} * [\mathbf{a}] + \boldsymbol{\varepsilon} * \boldsymbol{\delta}$

Further Techniques for Computation

Converting Representations [w]* →

[w]Preprocessing provides [**r**], [**r**]* for random **r**.

Open [**w**]*-[**r**]*, add **w-r** to [**r**].

Reorganizing bits between layers

- see paper for details

Techniques for Optimizing Complexity

To open values, send shares to one player, he reconstructs locally, does encoding if needed and sends result to all players.

Techniques for Optimizing Complexity

Players need to check that the opened value is in C (or C^*). We have a technique for checking that n vectors are codewords in time $O(n^2)$ with error prob $2^{-\Omega(n)}$

Actually, this is a new algorithm that can verify Boolean matrix product in time $O(n^2)$.

Output phase

1. Players stop just before output and commit to
   - Shares of MACs on all values partially opened so far

     (Actually a random linear combination of them)
   - Shares of values and MACs of final output
2. Open **α**
3. Players open first set of commitments and check MACs
4. Players open shares of output value/MAC and check

Conclusion

- A protocol in the preprocessing model for securely computing Boolean Circuits.
- Data, Computation and Communication overheads essentially $O(1)$.
- Linearly homomorphic MACs based on good codes with extra multiplication property.
- New algorithm that can verify Boolean matrix product in time $O(n^2)$ with error probability $2^{-\Omega(n)}$.